

CS101- Algorithms and Programming I

Lab 04

Lab Objectives: `while` loops

- For all labs in CS 101, your solutions must conform to these [CS101 style guidelines](#) (rules!)
 - Create a Lab04 workspace (i.e. the folder H:\private\cs101\lab04). This assignment has parts a, b, c, and d, each of which should be placed in a separate project within the same Lab04 workspace. Note: only one project is active at a time. To work (Build/Run) a different project, right click on the project's name and select "Set as active project".
 - You can only use `while` statements for this lab assignment. You cannot use any other repetition statements. You are not allowed to use arrays or any other data structures to store user input.
-

- a. Create a new project Lab04a. Write a program that takes an integer, `n`, from the user and calculates the largest perfect square less than `n`. A perfect square is an integer that is the square of an integer `x`; in other words, it is the product of some integer with itself. Display the result on the screen. Also, print the number `x` for each perfect square. For example, if the user entered `n = 64` the computer should give the result as 49 and 7, because the largest perfect square less than 64 is 49 and it is the square of 7. You should validate the user input as shown in the sample runs.

Note: You are not allowed to use Math class functions to solve this problem.

Sample runs:

```
> run Lab04a
Enter a positive integer n >= 2: 1
The input n should be positive > 1. Exiting...
> run Lab04a
Enter a positive integer n >= 2: 2.5
The input n should be an integer. Exiting...
> run Lab04a
Enter a positive integer n >= 2: 64
The largest perfect square less than 64 is 49 ( 7 ^ 2 )
> run Lab04a
Enter a positive integer n >= 2: 25
The largest perfect square less than 25 is 16 ( 4 ^ 2 )
> run Lab04a
Enter a positive integer n >= 2: 250
The largest perfect square less than 250 is 225 ( 15 ^ 2 )
> run Lab04a
Enter a positive integer n >= 2: 444444
The largest perfect square less than 444444 is 443556 ( 666 ^ 2 )
> |
```

- b. We'll say that a string is xy-balanced if for all the 'x' chars in the string, there exists a 'y' char somewhere later in the string. So "xxy" is balanced, but "xyx" is not. One 'y' can balance multiple 'x's. Create a new project Lab04b. Write a program that prints "This string is xy-balanced." if it is xy-balanced. If not, print "This string is not xy-balanced." See sample runs below.

Note: You are not allowed to use `indexOf()` and `lastIndexOf()` methods. You are expected to use while loop.

Sample Runs:

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is not xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is not xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is not xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is xy-balanced.
```

```
> run Lab04b
```

```
Please enter a string:
```

```
This string is not xy-balanced.
```

```
>
```

c. Create a new project, Lab04c, and inside the main method, do the following tasks. For these task, you will input a positive integer n from the user. You should validate the input. If the user enters a nonpositive value for n , keep asking for the input until the user enters a valid input. Use a single named constant to separate the following 4 parts in the output.

1. Write a while loop that outputs (separated by spaces, five numbers per line) the squares of the odd values from 1 up to n inclusive.
2. Write a while loop that outputs (separated by spaces, five numbers per line) the values which are divisible by 3 or 4, but not both, from n down to 3 inclusive. For example, if n is 12, it should display 9 8 6 4 3.
3. Write a while loop that outputs (separated by spaces, five numbers per line) $1 / n$, which are greater than 0.01, for every integer from n down to 1 with two significant digits. For example, if n is 5, it must output 0.20 0.25 0.33 0.50 1.00.
4. Write a while loop that outputs all of the divisors of n . For example, if n is 12, it must display 2 3 4 6 12.

Sample Run:

```
> run Lab04c
Please enter a value for n: 
```

```

----- 1 -----
1      9      25      49      81
121    169    225    289    361
441    529    625    729    841
961    1089   1225
----- 2 -----
33     32     30     28     27
21     20     18     16     15
9       8      6      4      3
----- 3 -----
0.03    0.03    0.03    0.03    0.03
0.03    0.03    0.03    0.04    0.04
0.04    0.04    0.04    0.04    0.05
0.05    0.05    0.05    0.06    0.06
0.06    0.07    0.07    0.08    0.08
0.09    0.10    0.11    0.13    0.14
0.17    0.20    0.25    0.33    0.50
1.00
----- 4 -----
2 3 4 6 9 12 18 36
> |
```

d. Create a new project Lab04d.

1. Write a program that inputs a sequence of integers from the user until the user enters an input which is not an integer and does the followings:
 - Display all adjacent pairs who are co-prime. In number theory, two integers a and b are said to be co-prime if the only positive integer that divides both of them is 1. Consequently, any prime number that divides one does not divide the other. This is equivalent to their greatest common divisor (gcd) being 1. For example, if the input is 3 3 5 5 5 4 7 7 4 6 2 2 3 3 3 8 0.5, the program should display
"Co-prime Pairs:
3 - 5
5 - 4
4 - 7
7 - 4
2 - 3
3 - 8".
 - Display the maximum and minimum of the inputs.
 - Display the sum and average of the inputs.
 - Display the count of the inputs.
 - Display the count of the inputs who are divisible by 7.

Note: Do not store all values read from the user; simply process each one as it is read!

Note: You are not allowed to use Math class methods to solve this problem.

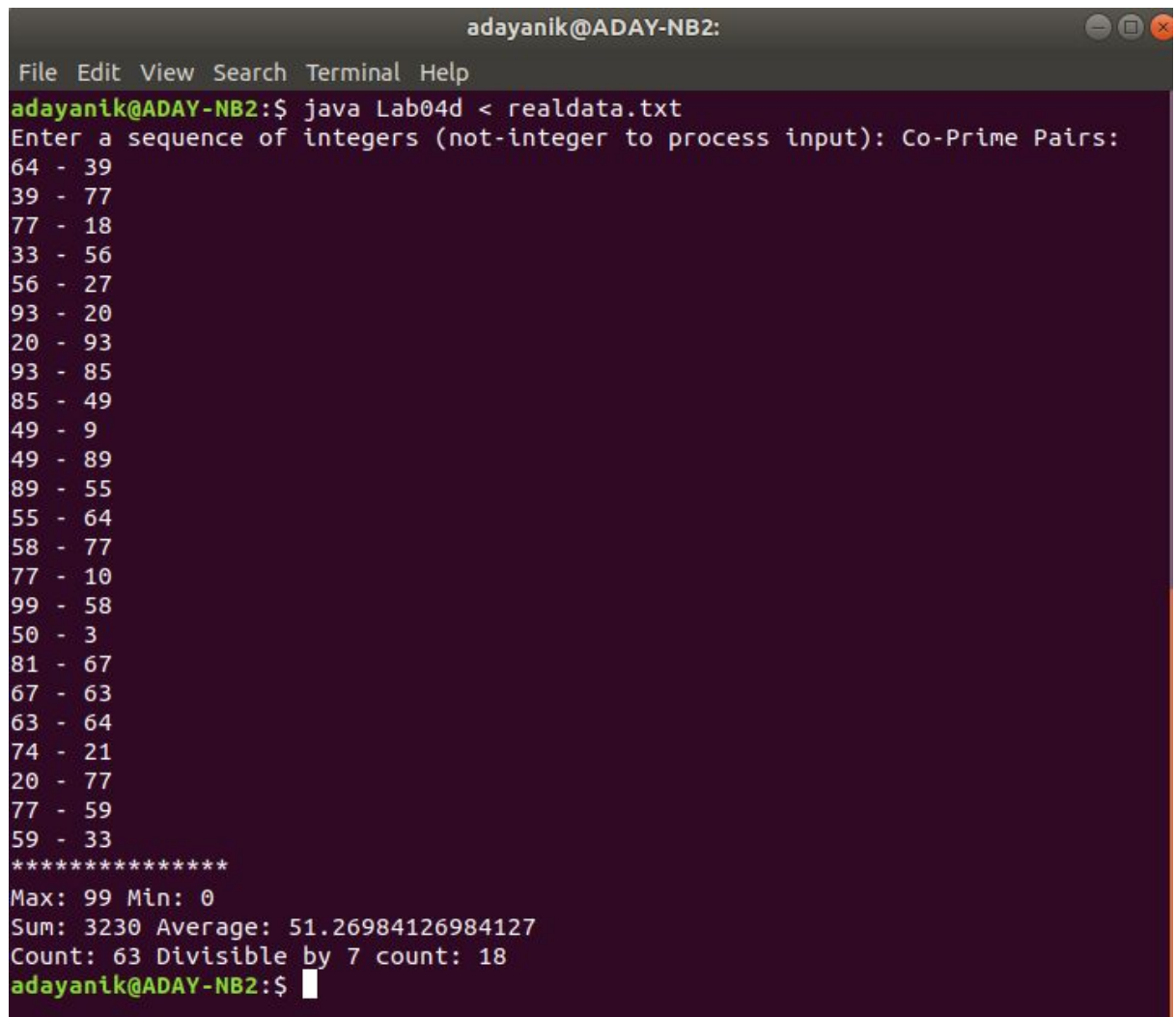
Sample run:

```
> run Lab04d
Enter a sequence of integers (not-integer to process input): 
No values entered.
> run Lab04d
Enter a sequence of integers (not-integer to process input): 
No adjacent pair is co-prime...
*****
Max: 28 Min: 2
Sum: 110 Average: 13.75
Count: 8 Divisible by 7 count: 3
> run Lab04d
Enter a sequence of integers (not-integer to process input): 
Co-Prime Pairs:
6 - 5
5 - 9
9 - 8
12 - 65
65 - 77
7 - 11
11 - 13
*****
Max: 77 Min: 3
Sum: 223 Average: 17.153846153846153
Count: 13 Divisible by 7 count: 2
>
```

2. In the same folder as your .class file, download and save realdata.txt text file. Examine the file. Notice that there is one value per line. Then run your program reading input from the given file `realdata.txt`. Follow the steps below to do it. See the sample run.

Open a command prompt and navigate to the folder containing your class file. After compiling your program from DrJava, from the command prompt, run your program by typing the command `"java Lab04d < realdata.txt"`, which should cause your program to read its input from the specified file, rather than the keyboard. It will still output its results to the command prompt. Note: you can have your program read data from one file and send the output to another file by typing `"java Lab04d < realdata.txt > results.txt"`. Try it, then open the output file `results.txt` to see what happened!

Sample run:



```
adayanik@ADAY-NB2:
File Edit View Search Terminal Help
adayanik@ADAY-NB2:$ java Lab04d < realdata.txt
Enter a sequence of integers (not-integer to process input): Co-Prime Pairs:
64 - 39
39 - 77
77 - 18
33 - 56
56 - 27
93 - 20
20 - 93
93 - 85
85 - 49
49 - 9
49 - 89
89 - 55
55 - 64
58 - 77
77 - 10
99 - 58
50 - 3
81 - 67
67 - 63
63 - 64
74 - 21
20 - 77
77 - 59
59 - 33
*****
Max: 99 Min: 0
Sum: 3230 Average: 51.26984126984127
Count: 63 Divisible by 7 count: 18
adayanik@ADAY-NB2:$
```

Note: The last sample run was taken on Ubuntu. The command window on Windows or Mac will look slightly different.